Fall 12-11-2019

3D Shape Prediction on Convolutional Deep Belief Networks

Gregory Y. Enriquez

Fall 2019

3D Shape Prediction on Convolutional Deep Belief Networks

Gregory Y. Enriquez

San Jose State University

3D Shape Prediction on Deep Belief Networks

A Project Presented to Dr. Robert Chun and

The Faculty of the Department of Computer Science

at San Jose State University

In partial fulfillment of the

requirements for the class

CS 298

By

Gregory Y. Enriquez

Fall 2019

The Designated Project Committee Approves the Project Titled

3D Shape Prediction on Convolutional Deep Belief Networks

By

Gregory Y. Enriquez

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

Fall 2019

Dr. Robert Chun, Department of Computer Science

Dr. Melody Moh, Department of Computer Science

Dr. Chris Pollett, Department of Computer Science

Abstract

The field of image recognition software has grown immensely in recent years with the emergence of new deep learning techniques. Deep belief networks inspired by Hinton [11] were one of the earliest methodologies of deep learning in the late 2000s. More recently, convolutional neural networks have been used in deep learning techniques, architecture, and software to identify patterns in imagery in order to make predictions such as classification, image segmentation, etc. Traditional two-dimensional, or 2D, images stored as picture files, typically contain red, green, and blue color data for each individual pixel in the picture. However, more recent commercial 2.5D or depth cameras have become more readily available such as the Microsoft Kinect, which is capable of capturing both RGB and depth (RGB-D) data. With the new depth dimension that can be captured from these cameras, objects are no longer limited to a flat dimension and the volumetric shape of the object can now be used to aid in recognizing that particular object.

In this project, I will utilize a convolutional deep belief network in order to observe the effects of rotation and sliding window stride when conducting classification on 3D models. An early study conducted named 3D ShapeNets experimented with this idea utilizing 3D computer aided design (CAD) model data in order to classify 3D models [2]. Extending from this research, the results from my research experiment showed an adverse correlation between angle granularity and recognition accuracy. Moreover, in regards to sliding window stride length, the training time increased substantially but had little effect on overall 3D model classification.

Acknowledgements

I'd like to thank Dr. Robert Chun for guiding and supporting me on this project and allowing me the freedom to explore a topic that I have a passion for. Additionally, I'd like to thank him for taking time out of his schedule to help and providing insight for whenever I needed it. Moreover, I'd also like to thank him for being a very influential professor in my parallel programming course leading me to learn more about CUDA parallel architecture which eventually led towards the field of deep learning.

I'd like to Dr. Melody Moh for taking time out of her schedule to be part of my project committee. Dr. Moh has always been very kind and supportive of my school work on top of being a very influential professor.

I'd also like to thank Dr. Chris Pollett for taking time out of his schedule to be part of my project committee. Dr. Pollett has always provided valuable insight and instilled discipline in me through hard work in my information retrieval course.

Lastly, I'd like to thank my girlfriend, my friends and my family for being patient and supportive while I pursued my Master's degree, balancing work and a hectic schedule at the same time.

Table of Contents

List of Figures

List of Charts

List of Tables

1. Introduction

The challenge that lies in the detection and classification of 3D objects then lies in the fact that a third dimension, depth, now has to be considered. Whereas 2D image detection and classification relied on the two-dimensional plane with pixels, 3D CAD models contain volumetric information for pixels which are more commonly known as voxels. This, in turn, significantly increases the size of the input vector for training [18]. Deep Belief Networks (DBN) have been shown to work well with smaller images but have shown to be relatively limited when working with much larger images [11]. Convolutional neural networks in the application of image detection and classification is a topic that has been researched extensively and scales well with larger images. It would then be a natural evolution to explore a relatively new hybrid neural network topology known as a Convolutional Deep Belief Network (CDBN) [2]. A hybrid Convolutional Deep Belief Network may be a promising candidate in that it could deal with the larger sizes of the neural networks. In this project, I will utilize an existing research project with a CDBN framework and assess the accuracy of the network while manipulating incremental view angles and filter stride distance.

Artificial Intelligence (AI) and Deep Learning have quickly grown with the emergence of powerful and cost-effective graphics processing units (GPU). While originally purposed solely to transform pixel data information to a visual display, GPUs have been found to be superbly useful in Deep Learning. Modern GPUs possess the necessary hardware to support massively parallel processing applications such as Deep Learning, containing many nodes in graph like structures known as neural nets [13]. While not to be confused with cores in central processing units (CPU), GPU cores are relatively simple arithmetic logic units (ALU) capable of performing

small and simple calculations. Central processing units have ALU components as well but possess less as CPUs are capable of performing much more complex and general actions.

Prior to the inception of deep learning, artificial intelligence efforts were primarily focused on programming a computer to reactively perform actions based on a certain input or criteria. This branch of artificial intelligence is now more modernly known as narrow AI, as the programming is defined to work on specific tasks rather than a more generalized approach. Deep learning, on the other hand, is a more broad and general approach towards artificial intelligence and can be applied to different fields such as medical research, autonomous driving, pattern and image recognition, weather prediction and more. On a lower level, deep learning is capable of working in these different domains by the utilization of neural networks which are used to identify patterns similar to how the human brain interpret patterns. The model of neural networks has stemmed from the infrastructure of the human brain whose billions of neurons are used to observe, learn, and categorize information.

A particular area of research where deep learning has begun to perform very well is in the field of computer vision and image recognition. By using convolutional neural networks (CNN), image recognition software is now able to identify patterns in 2-dimensional (2D) images, learn from them, and make certain predictions about them [8]. For example, after being shown thousands of images of a particular object, such as a dog, a CNN could then be shown a brand new image of a dog and classify whether or not it, in fact, is an image of a dog with a certain degree of confidence. The structure of CNNs may vary but many computer vision configurations contain convolutional layers and fully connected layers. The convolutional layers entwine or convolve input and typically pool them reducing dimensionality and pass them as

input to another convolutional layer or pass them into fully connected layers which perform the recognition tasks. Convolutional neural networks work by creating multiple network layers which, in turn, granularly filter over very small subsections of an image, applying a convolution filter over the area and then generates a final weighted sum calculation [8]. This operation is performed a very large number of times within the full resolution of the image and at the end classifies the image or detects an object based on the output result.

Object detection is another important area of image recognition and computer vision. Unlike the example of the dog image stated previously, some neural networks may be smaller in scale and only be responsible for detecting objects such as in embedded systems for self-driving cars in real-time. Although self-driving cars may need to classify a particular object it sees, it also needs to be very aware of the many objects in its surroundings. If the objects are not required to be classified, typically 2D boundary boxes are superimposed on images when identifying objects in images in order for the human eye to validate detection and in some cases make use of the information [1]. In this case, classifying objects themselves may not necessarily be as important as identifying that there are in fact objects that are obstructing the vehicle and whether or not that object happens to be moving. Most importantly, in the application of self-driving cars, the identification of humans and animals need to be both fast and accurate in object detection.

As self-driving cars are operating in 3D space, it is required that the computer in the car track many objects simultaneously while determining whether or not objects are moving. Sensors and self-driving car cameras are only capable of viewing objects from a 2D perspective, despite being in a 3D space, which begs the question of to what degree must computer vision

account for in determining the position and rotation or the pose of an object. One method in aiding to differentiate objects, say from the driving road, is the usage of depth sensors. These depth sensors, in most cases, project lasers outwards which reflect and collect distance information for anything that they collide with, partially reconstructing the environment in a virtual space. The data produced from depth images can be extracted to produce grayscale images identifying different distances by grayscale color values [17]. For example, objects the same distance away from a sensor would possess similar if not identical grayscale color values.

While conducting research in 3D space, it is natural to look to computer-aided design (CAD) models. The reasoning for this is that typical 2D images are flat and do not offer any other volumetric environment data or anything about hidden or occluded components of objects. And although RGB-D images do provide depth information, like 2D images, they do not offer any data about occluded components as the 2D perspective is static. CAD models, on the other hand, may be synthetic recreations but do contain full 3D volumetric data albeit with an overall lower resolution. Modeling tools provide the ability to change the perspective or viewport by rotating around the CAD model. Therefore, given more processing time, multiple perspectives and object rotation, CAD models may contain much more valuable information in classifying the complex objects that may contain parts that are occluded from view [2].

In the process of the classification of 3D objects, there is the third dimension of depth that must be accounted for. Combining volumetric information from different angles, or poses, of 3D models would provide additional data that the model possesses while also substantially increasing required training time. Therefore in this research experiment, I will examine the effects of modifying the angle in which the models are captured in order to see if there is an

effect on recognition accuracy. In Section 2, background information will be discussed regarding the topography of convolutional neural networks, the behavior of the internal layers, and the properties of 3D models such as volumetric data that is used as input. Lastly, Section 2 will close on discussing deep belief networks and why the hybrid topography of a Convolutional Deep Belief Network would be advantageous in 3D model classification.

Following background research of this project, Section 3 will discuss the current state of the art for experiments that have been conducted in 3D model classification. Additionally, the technology and current state of software used to create neural networks will be outlined. Next, the Section 4 will discuss the experiment hypothesis and testing methodology that will be used in this project. Section 4 will list steps taken to reproduce as well as the requirements for the project. Section 5 will list the project tools and hardware configurations that were used for the experiment. Section 6 will list the test configurations that were ran as input and Section 7 will provide the results as well as the analysis of those results.

2. Background

The following section will discuss the concept and components of convolutional networks, deep belief networks, image recognition with neural networks and finally the current state of 3D object detection and classification.

2.1 Convolutional Neural Networks

Convolutional neural networks in regards to image recognition rely on utilizing several different layers [3]. The first set of layers are the convolutional layers which act by applying many different filters over the image moving a given number of steps each time. Following is the pooling layer, whose primary task is to reduce the overall dimensionality of the image by taking the output of the convolve step and grouping them into single values. Lastly, the fully connected layer is tasked with taking the output of the pooling layer and seeing if that output fits to a pre-trained labeled image.

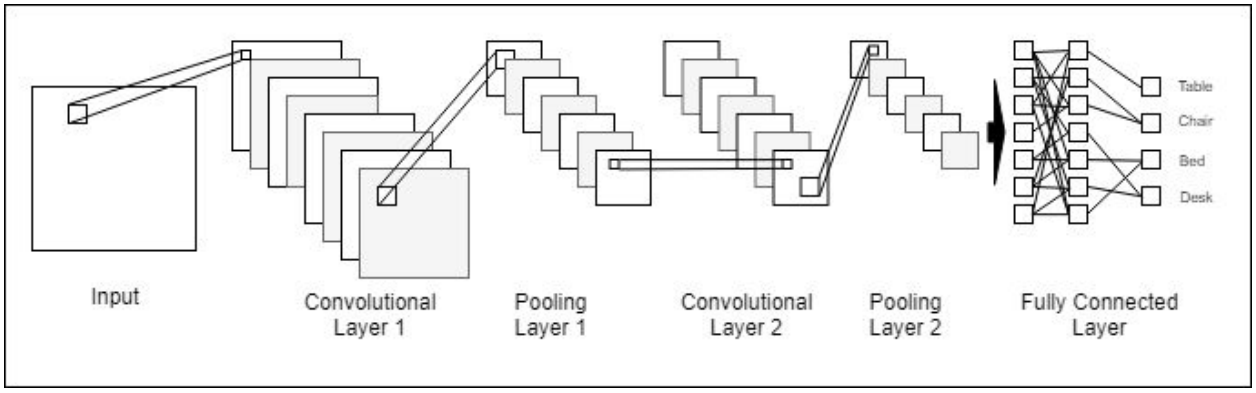


Figure 1: Layers of a Convolutional Neural Network

2.1.1 Convolutional Layers

For image recognition through deep learning, the role of the initial visible convolutional layers is to apply a filter or a convolutional mask on top of the original image, generally starting from the top-left most position and taking steps or strides from left to right, top to bottom. An example of a convolutional filter may be segments of a figure such as small curves, lines with different orientations, corners or even loops and are typically determined by a phase called feature extraction as mentioned in Section 2.2. As the filter traverses the 2D image, it groups the parts that it was applied over and outputs a value for that segment based on how closely it matched the filter. The result is an output of how well the filter matched a particular area which may or may not activate the following node. The outputs of this layer are sometimes put through intermediate logistic or sigmoid functions which work to normalize values in the output and either put them in the range of 0 or 1, with 1 indicating a partial or complete match. The equation for a standard sigmoid or a logic function is as follows:

$$S(x) = \frac{1}{(1+e^{-x})}$$

and whose curve is shown below for all values of x. As the graph shows, all output values

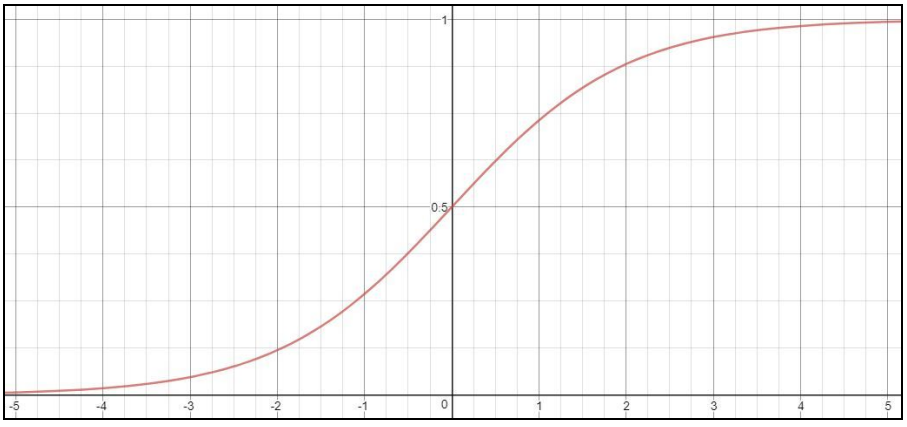


Chart 1: Sigmoid or Logistic Function Plot

range between 0 and 1, and this helps to deal with the values of each of the individual weights that each neuron possesses. Optionally, other networks may apply a rectifier linear unit (ReLU) instead which similarly normalizes values to a range of 0 and 1 but instead only takes into account positive values for outputs. What this means is that negative values, which is sometimes correlated with empty or negative space, is simply zero. The equation for a basic ReLU is

$$S(x) = max(0, x)$$

and a graph of what a ReLU looks like is shown below. The difference between a ReLU and a standard logistic function is that positive values can exceed the output of 1 and now the strength of how well a particular filter and an input image is unbounded. An issue known as the

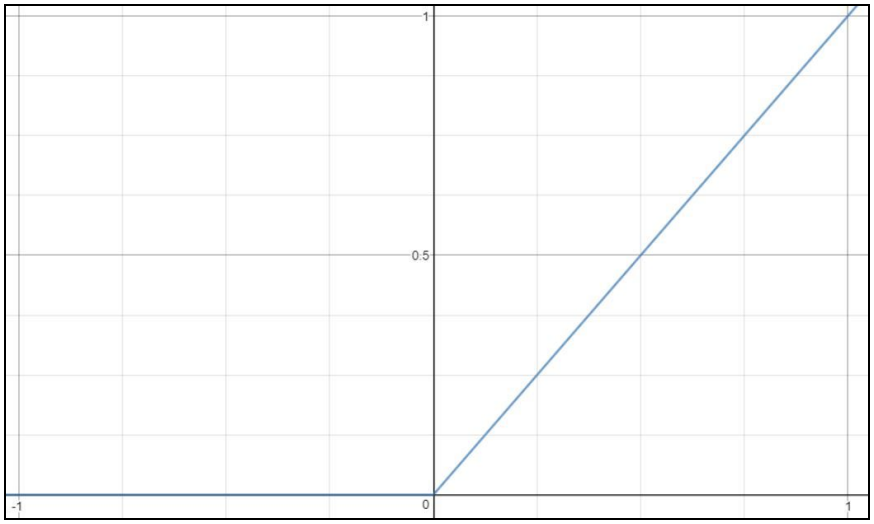


Chart 2: Rectifier Linear Unit (ReLU) Plot

vanishing gradient can occur with small values, so in some cases another type of activation function is considered known as a softplus function. The softplus function is as it sounds, and softens the ReLU output to gradually increase as the input moves away from the zero values. A basic type of softplus equation is shown below followed by its plot. Multiple

$$S(x) = log(1 + e^x)$$

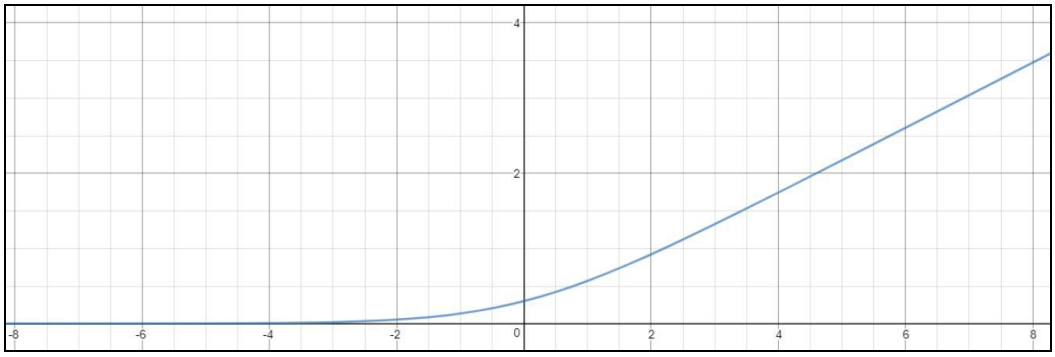


Chart 3: SoftPlus Plot

convolutional layers can be present depending on the network topology and the size of the image.

One important role that the convolutional layer plays is that it works to produce a smaller dimensional array that is smaller than the original, depending how large of a filter and how many strides were used. For example, if a 2x1 filter is applied to a 10x1 image using a stride size of 1, the resultant array would be of size 9x1 as the filter would start from the first position and take steps of size 1 to the right until the end was reached. Another example would be given the same input size but a stride of two, the resultant array would be of size 5x1. The results are visual representations of the original image into smaller pieces, which may or may not be easily recognizable from the human perspective. These smaller pieces are effectively the visual output of how often the filter pattern was seen while propagating through the image data.

2.1.2 Pooling Layers

After the convolutional and ReLU layers, the output then acts as the input into the pooling layer. The purpose of the pooling layer is to group previous layer output values of the convolutional layer and group or pool them together. Whether nodes in the pooling layer contribute to different parts of the image depends on whether the previous neurons in the convolutional layer were activated. For example, if there was a filter in the convolutional layer

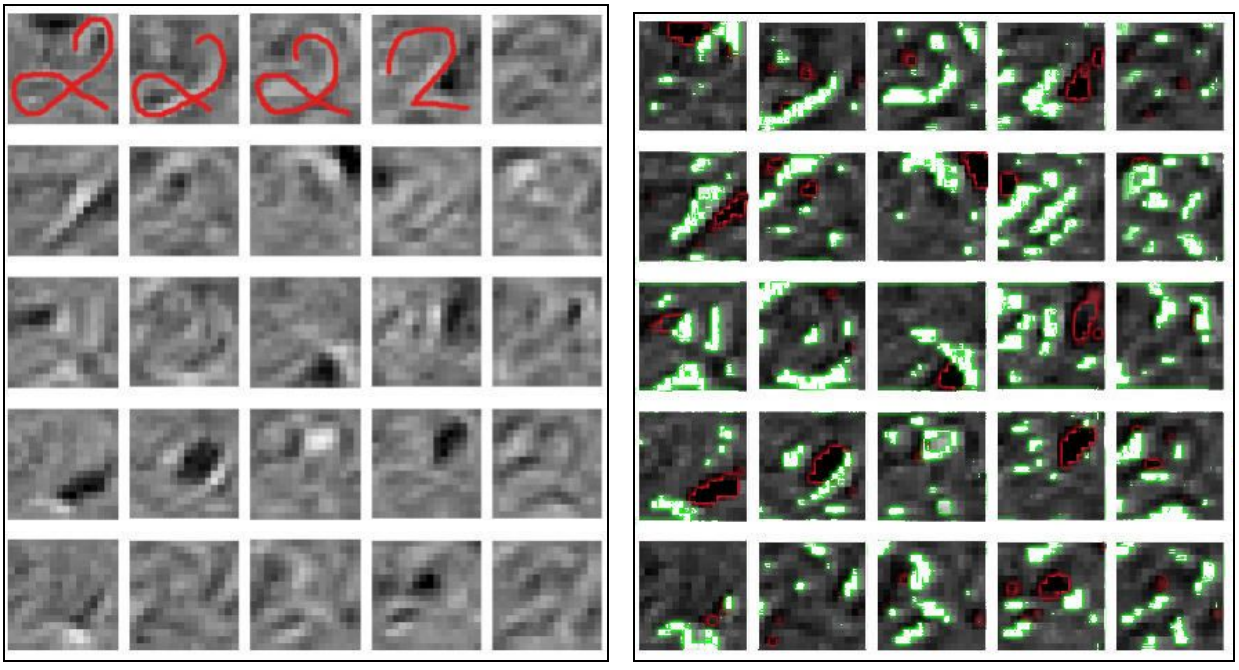
that was an "L" shaped segment within a 2x2 segment, the convolutional layer would apply this "L" shaped filter across the entire image and the pooling layer would partially output an array indicating how many and in what areas the filter activated a neuron. Likewise, all other filters are applied in the same way and group this information which is why the number of overall dimensions is reduced. This is possible since the number of dimensions is not based on the raw number of pixels but is instead based on whether specific patterns emerge in different parts of the image. This aggregate of information is all combined and then output to the fully connected layers which then are responsible for applying existing, labeled filters to determine if a particular object was found within the image.

2.1.3 Fully Connected Layers

The last layer, the fully connected layer, is tasked with comparing the aggregated result from the pooling layer with existing labeled data when the network was trained. During training, labeled data will traverse through the network and be broken down into segments of the overall composition of the data. As this labeled data passes through the network, specific paths through the network are activated and in the final layer the classification of the object is stored, a filter is created, and the network is then trained in recognizing that particular labeled input. It is important to note that during the convolutional steps, because the image gets broken down into smaller pieces, objects may potentially appear in different locations in the original image. What this means is that the recognition of objects in new images do not necessarily need to be in the same location as an object in a previously recognized location. This is one of the most critical aspects of object recognition in deep learning.

2.2 Deep Learning Object Detection

Features of an object or an image that humans would normally recognize such as curves, lines, loops, and general shapes are not extracted the same way by computer neural networks [11]. The human brain and visual cortex has been theorized to identify objects by the grouping of primitive shapes such as cylinders, blocks, etc. known as Gestalt principles. Computer neural networks perform similar actions when recognizing objects; however, common shapes are not recognized, and instead, patterns that are recognized appear almost random as shown in Figure 2 (b). The general shapes of the handwritten two were drawn over in Figure 2 (a) to highlight full patterns. In Figure 2 (b) the areas in which the computer itself saw patterns were overdrawn with increased contrast.



(a) (b)

Figure 2: Deep Belief Network Feature Extraction Example
for (a) the Number 2 Handwritten [11] with (b) Highlighted Activations

Also the negative weighted areas were surrounded with red while positive features were surrounded by the color green. The description of the extraction of features and the creation of filters in the following section represents a more understandable approach; however, it is simplified and not realistic.

In relation to 2D object detection and convolutional neural networks as mentioned in the previous section, features are extracted from images during the training phase and create object image filters. An example of this would be a shape such as a star in Figure 3 where the network cuts the overall shape of a star and extracts the features of the star that it discovers. In this case, the star would be broken down into different features that the star possesses such as lines and different types of joint line segments connected together by a variety of different angles. Figure 3 below, portrays filter creation in a simpler illustration, but in actuality, the features are broken down even more to individual line segments and so on. During the convolution phase, each of these feature patterns are run against the test image to produce multiple filtered images which will then be input to ReLU and pooling phases. These features presented, whether from overall shape of the figure or by more granular features, are used together are to create a hierarchical feature structure used in deep neural networking [6].

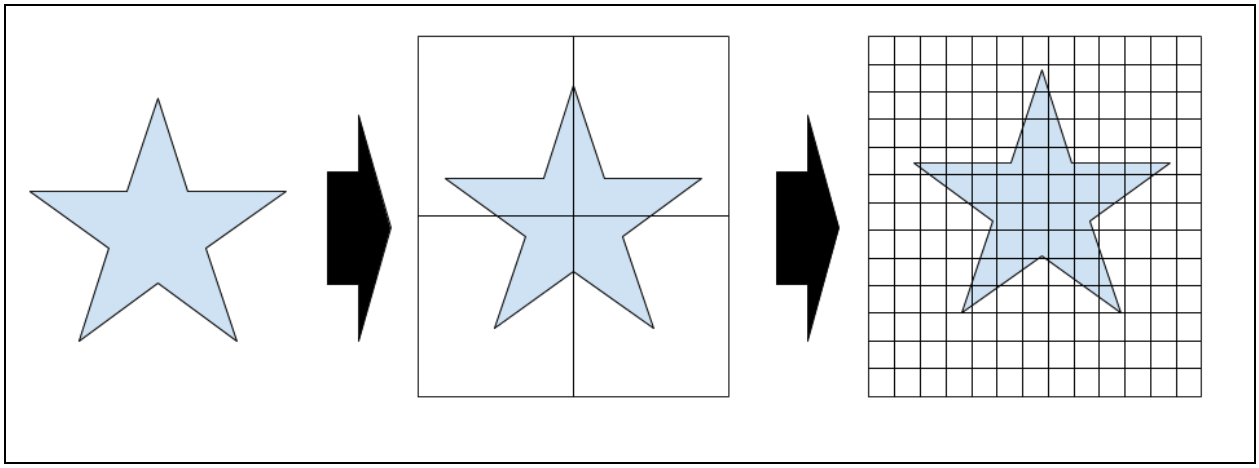


Figure 3: Simplified Convolutional Feature Extraction

2.3 3D Deep Learning Object Detection

Similarly, research has been conducted in the areas of the development of convolutional neural networks (CNN) for 3D modeling. As 2D images use features composed of different pixel arrangements in their patterns in their CNN during the convolutional phase, 3D models use binary voxel values, either filled or unfilled, in a similar fashion [4]. The dimensionality for 3D patterns is increased by one as depth information is taken into account. The overall size of the network is therefore increased, as well as the computer memory required to hold it. For 2D shapes with the smallest multi-pixel arrangement being a 2x2 rectangle, the number of possible patterns is 8, but for a voxel 2x2x2 cube composition it would be 256. The 3D ShapeNets project is modeled from a well known CNN architecture, AlexNet, which was introduced in 2012 and used for 2D image recognition [4]. One of the primary differences that 3D ShapeNets CNN's do is place more of a focus on shape and geometrical features rather than putting an emphasis on object colors [2]. This would support existing research that suggests that texture

pattern and colors play a smaller role in 2D image recognition which may carry over to 3D shape recognition [3].

One primary difference of using a Convolutional Deep Belief Network versus other neural network architectures is the use of Restricted Boltzmann Machines (RBM) [11]. Restricted Boltzmann Machines are known for the ability to accurately reconstruct the previous or visible layer. RBMs score the reconstruction using a method called contrastive divergence which in turn helps to minimize the error in assigning the weights and biases created through the process. Conversely, classical neural networks such as Multilayer Perceptron (MLP) networks use methods like back propagation which have been known to cause training problems such as the vanishing gradient. RBMs do not run into this issue as visible and hidden layers are shallow and fine-tuned before outputting data into the following layers.

A secondary advantage of a Convolutional Deep Belief Network is the benefit of the convolving step which aids in two primary ways. The first way the convolving approach helps is to recognize objects independent of where they may be on a particular image since the feature filters that are created traverse the entire image. MLP networks on the other hand require additional training data to be present if an object may appear in different locations of the input data. The secondary benefit of the convolution step is that neurons between layers do not need to be fully connected which reduces processing requirements and unnecessary connections within the network topology.

2.4 CAD Models

Computer-aided Design Models (CAD) have been used in many different applications for the purposes of designing and prototyping 3D models. As these models are composed of three

dimensional values, namely: length, width, and depth, their unit of measurement in 3D modeling space are known as voxels, which is analogous to pixels for 2D images. These voxels in these 3D models are also similar to a 3-dimensional array whose values are binary as the voxel space is either filled or empty.

As mentioned previously, 3D CAD model datasets have significantly less training data available than their counterpart 2D sets which have grown over many years [2]. Other research has suggested that the solution for this problem is to use video capture data which does not necessarily require CAD labeled data. Some research suggests that video capture data allow the training to process the video frame by frame while following separate objects in the video in order to build a 3D model of the object [5].

2.5 Deep Belief Networks

One early application of utilizing deep learning techniques was handwritten number recognition by Hinton [11]. In his research and lectures, Hinton illustrated that neurons in a neural network worked by extracting features of handwritten numbers that we as humans do not necessarily understand. An unrealistic, simpler, and understandable example would be that we as humans recognize dashes, loops, or curves of handwriting that we put together to form a symbol that we eventually recognize as a number. Deep belief networks (DBN), generated from computers, extract features that may be practically any part or multiple parts of a handwritten number. One such example would be that many numbers include an arc at the top of the number such as in the cases of 2, 3, 6, 8, 9 and 0.

Deep belief networks are typically composed of stacking Restricted Boltzmann Machines (RBM). In comparison, Convolutional Neural Networks uses filters that traverse the object that

builds the connections between the previous layer and the following layer. Because of this, neurons in one layer of a CNN do not connect to every single neuron in the following layer, whereas in a DBN architecture, every neuron in the initial visible layer connects to every single neuron in the following hidden layer. In the graphical topology, a simple RBM consists of two sets of nodes that have undirected connections with one another with the *restriction* that no nodes in the visible layer are connected to any other visible nodes, and in the same fashion, no hidden nodes are connected to any other hidden nodes. The result is similar to that of a bipartite graph as shown below.

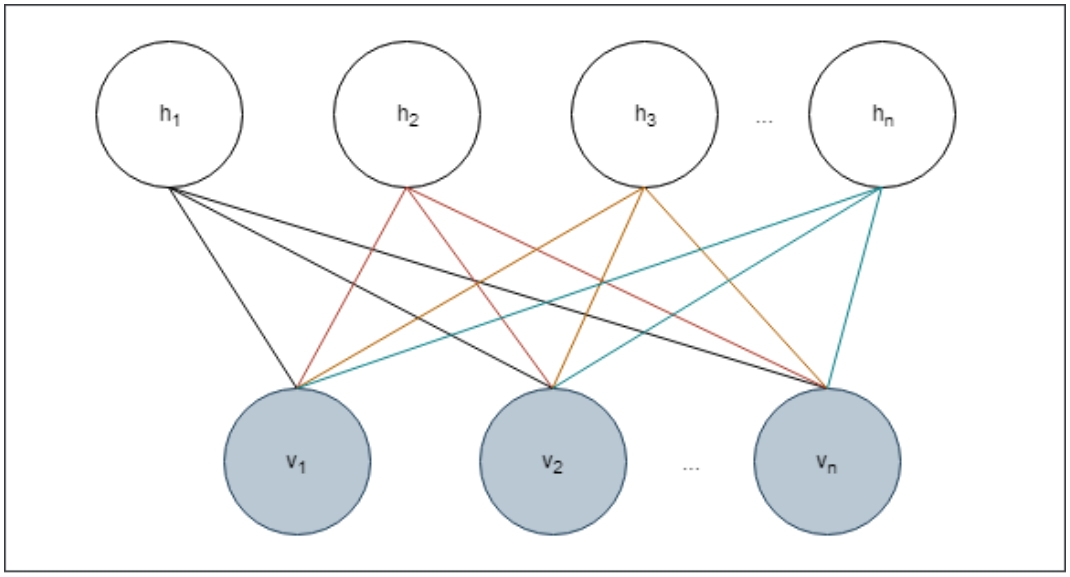


Figure 4: Basic Restricted Boltzmann Machine (RBM) Topology

The bottom set of nodes are the visible nodes while the top are hidden nodes. From a simpler perspective, RBMs work by taking a pair of two nodes, one visible and one hidden, and finding the next visible node's value based on an algorithm called Gibbs sampling. It goes through this process through potentially thousands of iterations until the RBM is trained to properly predict the outcome of a given input. For example, in the first iteration, the value of the first hidden node is based on random sampling given the conditional probability of the first visible node and

additional weights and biases which are also passed through an activation function. Whereas the next visible node has the probability of being activated by the outcome of the first hidden node that was just calculated. For the first iteration, the first hidden node and second visible node have equations as follows:

$$P(h_0|\, v) \;=\; sig(W_{v_0} \cdot v_0 \;-\; b_{v_0})$$

$$P(v_1|\, h) \;=\; sig(W_{h_0} \cdot h_0 - b_{h_0})$$

After continuously repeating this process, the RBM eventually sets its values to be roughly equal to a network that is able to reconstruct a specific data input. Each of the individual nodes in the RBM represent the activation of a specific feature inside of an image. As mentioned previously, a feature of an image may be a line, curve or even larger and more complex shapes. The RBM maps each feature to each of the visible units and activates it based on conditional probability and the likelihood that this particular feature is part of some input that was recognized before.

After this process is complete, a value is then generated by comparing the reconstruction to the original, and then adjusting weights in the association to attempt to recreate a better reconstruction on the next attempt. After following this process multiple times, the output of this is a hidden layer with close to or good enough values for weights and biases which then can be used for the next RBM. In a DBN, multiple RBMs have been shown to be effective in recognition of two dimensional handwritten characters as previously noted [11]. It is important to note that the traditional DBN topology does not reduce in later layers as there is no convolving step such as in CNNs. This means that every layer within a DBN is of equal size and thus require larger amounts of computer memory whereas CNNs have a large requirement for the first layer and reduced for each subsequent.

2.5.1 RBMs with GPUs

As mentioned above, the equation in calculating the next visible or hidden layer contains one critical piece of information: The subsequent hidden node is only dependent on the previous visible node and not any other hidden nodes. This is true vice versa. In effect, what this means is that the steps that the RBM takes can be done not just in parallel but in massively parallel systems such as GPUs where memory of the previous visible or hidden layer is stored and the subsequent layer is calculated. The reason for this is due to the *restricted* nature of an RBM where there are no visible-visible or hidden-hidden connections. The values for each of the layers is calculated independently--that is, the visible is calculated, then the hidden is calculated, then the visible is calculated again, etc. In terms of computation time, this illustrates that the network can be trained in O(N) time where N is equal to the number of the larger between the visible and hidden layers. Without this restriction in place, the time complexity would be in the order of $O(N^2)$ as each visible layer would have to wait on the previous visible layer calculation.

2.6 Convolutional Deep Belief Networks

In the previous sections, it has been noted that Deep Belief Networks have been used to train a network for the purposes of object recognition, and more specifically, in the classification of handwritten numeric digits [11]. However, the images used in the study were 28 x 28 pixels in grayscale which drastically reduces the overall size requirements of the DBN. The primary limitation of the DBN is that each of the sizes of the RBM layers are the same and do not reduce in size as data propagates through the network. This limitation is exacerbated in 3D models due to the third depth dimension which would even further expand the necessary requirements to

create a DBN. Despite this limitation, DBNs have been shown to perform very well in recognizing and recreating input data, and this may be adaptable to 3D models.

Although Restricted Boltzmann Machines are stacked in a typical DBN structure, RBMs have also been shown to perform well even when using just a single layer as the RBM can iterate using contrastive divergence until the layer is able to get close to recreating the original input. Contraversely, CNN have also been shown to perform well in recognition tasks but networks can often take a large amount of time to train and is subject to irregularities or a mistrained network. Although the network is capable of recognizing what it defines to be as features of a specific set of training data, as mentioned previously, CNN extract their features by subsections of the input images. One possible consequence is that a CNN may misplace the location of parts of an object such as eyes or ears of a human head. Therefore, RBMs would be a suitable solution to this problem in that training data can be used to teach the network whether it is correct or incorrect in recreating an input. Furthermore, CNNs in fact do reduce the size of progressive layers since convolutional filters are used which group together patterns that it recognizes in input images.

A hybrid design combining the convolutional nature of a CNN alongside the RBM structures of a DBN would therefore be plausible, and was in fact performed [2]. The study of 3D Shapenets will be discussed in the following sections. In Figure 5 below, a version of this design is presented where convolutional and pooling layers are in the early stages, and then followed by a fully connected layer that is joined with an RBM structure for generative training using contrastive divergence to ensure that the network can use its nodes to recreate an input.

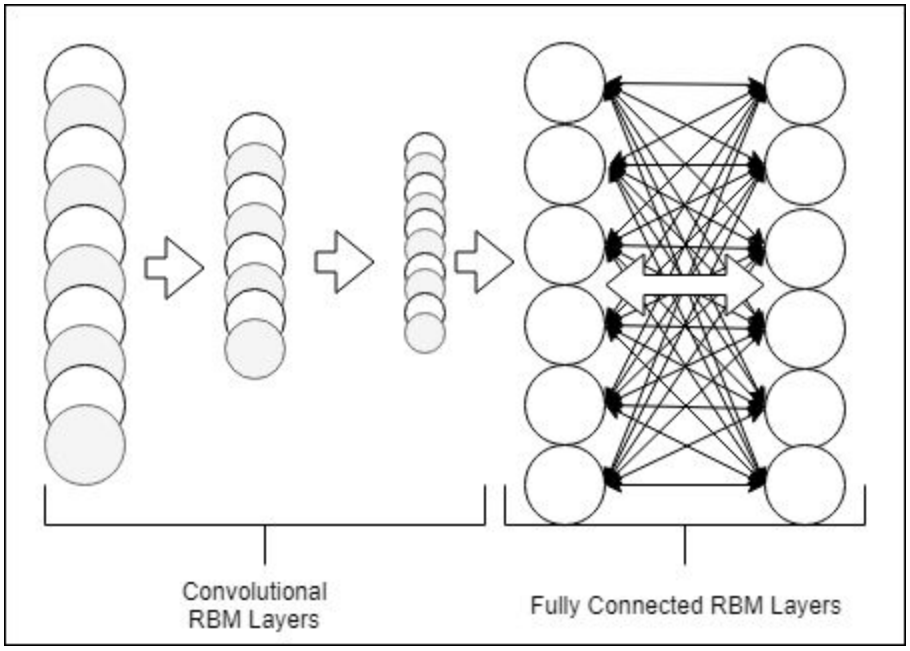


Figure 5: Simplified Convolutional Deep Belief Network

3. State of the Art and Related Works

The following section will analyze research that has been conducted in the areas of the usage of deep convolutional neural networks for 3D shape recognition along with the current technology that supports it.

3.1 3D ShapeNets

In 3D Shapenets, Wu et al. introduced a hybrid neural network structure they called a Convolutional Deep Belief Network (CDBN) [2]. The CDBN they illustrate in the paper, combine the feature extracting nature of a convolutional neural net (CNN) and the reconstruction feature of a deep belief network (DBN). In the first three layers of the network, as mentioned in Section 2.1, convolutional layers are implementing into breaking down the 3D model into smaller groups using 3D filters such as the one below which illustrate potentially two sides of a table meeting at a curved corner.

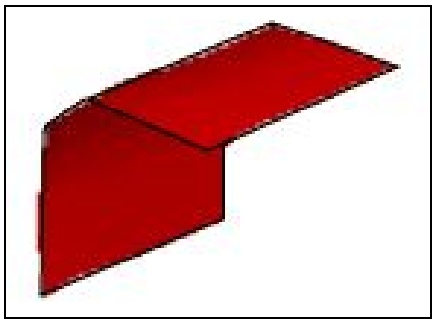


Figure 6: Example of 3D CAD Model Filter [2]

3.2 3D Object Detection in RGB-D Images

The current state of 3D object detection is fairly limited. As mentioned in the previous section, a limitation of the study was that the volumetric dimensions were very limited only using 30 x 30 x 30 voxel models. A beneficial outcome of the study was also the composition of

what the researched named ModelNet, which was a curated 3D model CAD collection from a larger CAD database.

Another study conducted around 3D object detection by Peng, et. al. in a convolutional neural network was also done which studied the behavior of more granular data within 3D models, namely pattern recognition with textures of models [3]. The study concluded that 3D convolutional neural networks were fairly invariant to surface detail for models and that the networks were much more sensitive to overall figure and edges around it. This fact would suggest that the problem of using low voxel resolution figures may not be a hindrance when using smaller 3D CAD models to train volumetric data within the network.

The study in which this paper was most interested in was a 2017 study named Deep Sliding Shapes which proposed the first 3D Region Proposal Network (RPN) [1]. This study performed the task of predicting 3D boundaries or regions for objects within RGB-D images training with 2D and RGB-D images. The study, however, opted to not train with 3D CAD models despite references to the 3D Shapenets study which is the primary area that I will explore. The study also proposed the first Object Recognition Network (ORN) which would be a convolutional neural network which concatenates two separate entry points, namely 2D and RGB-D convolutional neural networks, into a single one after a single fully connected layer [1].

Deep Sliding Shapes [1] mentions that many object detection schemes for imagery only illustrate a 2D bounding box. While this is informative in terms of overall object detection, 2D bounding boxes superimposed on an image does not really provide any other information, such as actual object boundaries as well as any sort of depth information of the object. Providing such information with dimensions including depth, may unlock additional possibilities in object

detected. One example of this the paper mentions is the use of 3D object boundary detection for robotics such as packaging robots who are reliant upon knowing the dimensions of a box before picking it up.

The joint Object Recognition Network (ORN) that the authors of Deep Sliding Shapes have created focuses on combining parts of two separate neural networks into one. This joining of two different neural networks suggests that it may be possible to also include 3D CAD imagery into the training phases of the networks in hopes that the CAD training may improve overall accuracy of the network in determining 3D dimensions and boundaries for objects within the RGB-D images.

3.3 NVIDIA CUDA SDK

Most if not all of these studies conducted using convolutional networks are run on GPU hardware as it utilizes parallel processing on a massive scale. The utilization of parallel processing has much to deal with certain phases of the convolutional neural networks (CNN). For the convolutional phase, the 2D or 3D arrays that are used allow individual indices to be mapped to arithmetic logic units (ALU) which GPUs can possess thousands of. As these operations in the convolutional phase are very small arithmetic or zeroing operations, these individual ALU's are perfect candidates as they are capable of performing the operation in parallel inside the GPUs unlike central processing units (CPU) which possess a much smaller amount of ALUs.

The company NVIDIA that provides these GPUs commercially, also contain many tools for creating neural networks on their deep learning platform which is programmed through the CUDA language. For the hardware that I will use for my experiment, I will be using an NVIDIA

GTX 1080 TI GPU, which is a relatively more affordable solution for GPU deep learning and is substantially cheaper than enterprise deep learning GPU configurations that cost many thousands of dollars. The GTX 1080 TI GPU is somewhat comparable to the hardware used in the Deep Sliding Shapes study which was a K40 GPU, released in 2013. Table 3 illustrates the primary features of each of the hardware setups. Given this table, I should have the appropriate hardware to conduct this experiment and expand upon it.

4. Test Methodology

The following section will describe the software requirements for the project as well as how to normalize the data from the 3D models to be processed. Afterwards, the steps to train and evaluate recognition accuracy will be listed in order to run the experiment.

4.1 Project Software Requirements

The software used for this project tried to best match the existing project, 3D ShapeNets, and used versions either listed explicitly or that were available at the time. A list is available in Table 1. The Ubuntu OS version used was 14.04 and the Matlab version used was 2013a. The g++ compiler used was version 5 and the CUDA toolkit version was 7.5. One exception had to be made in order to make the project run: The NVIDIA drivers used were the latest as the previous versions did not support the newer GPU used, the GTX 1080 TI. Therefore, the driver that was built into the CUDA toolkit was replaced by NVIDIA GPU driver version 381.

4.1.1 Project Software Requirements

Operating System	Ubuntu 14.04
Software Requirements	Matlab R2013a NVIDIA CUDA Toolkit 7.5 NVIDIA Driver version 381 g++ v5
Hard Drive Space Requirements	300 GB

Table 1: Project Software Requirements

4.2 Test Steps

The next few sections will cover the steps required in order to format the data, modify configurations, run test and output results.

4.2.1 Data Normalization

The first step is to normalize shape and vector data into 3D voxelated format. The reasoning for this is the shape and vector data may be of various lengths, widths, and depths. Additionally, as previously mentioned, the dimensions increase the amount of memory required by the network fast as there is the depth dimension. The shape and vector data is found in the *volumetric_data* directory. The Matlab command can be used to prepare the data into the 3D model format:

```
write_input_data(off_path, data_path, classes, volume_size, pad_size, angle_inc)
```

Arg	Description
off_path	Path to store output voxel data
data_path	Path where shape data can be read from
classes	List of different shapes (classes) to read
volume_size	Length of one side of the cubic figure
pad_size	Length of padding from any side of the square encompassing the model
angle_inc	Number of degrees to increment when rotating point-of-view camera of the model

Table 2: Configuration Options for Preparing Voxel Data

An example input and the one used for this project was:

```
> write_input_data('model_data', 'volumetric_data', {'bathtub', 'bed', 'chair',
'desk' 'dresser', 'monitor', 'night_stand', 'sofa', 'table', 'toilet'}, 24, 3, 30)
```

A visual example of this normalization of the models is shown below where the left illustration is the much larger mesh model composed of data points in OFF format. The right side is a rough estimation of how the voxel CAD model would look.

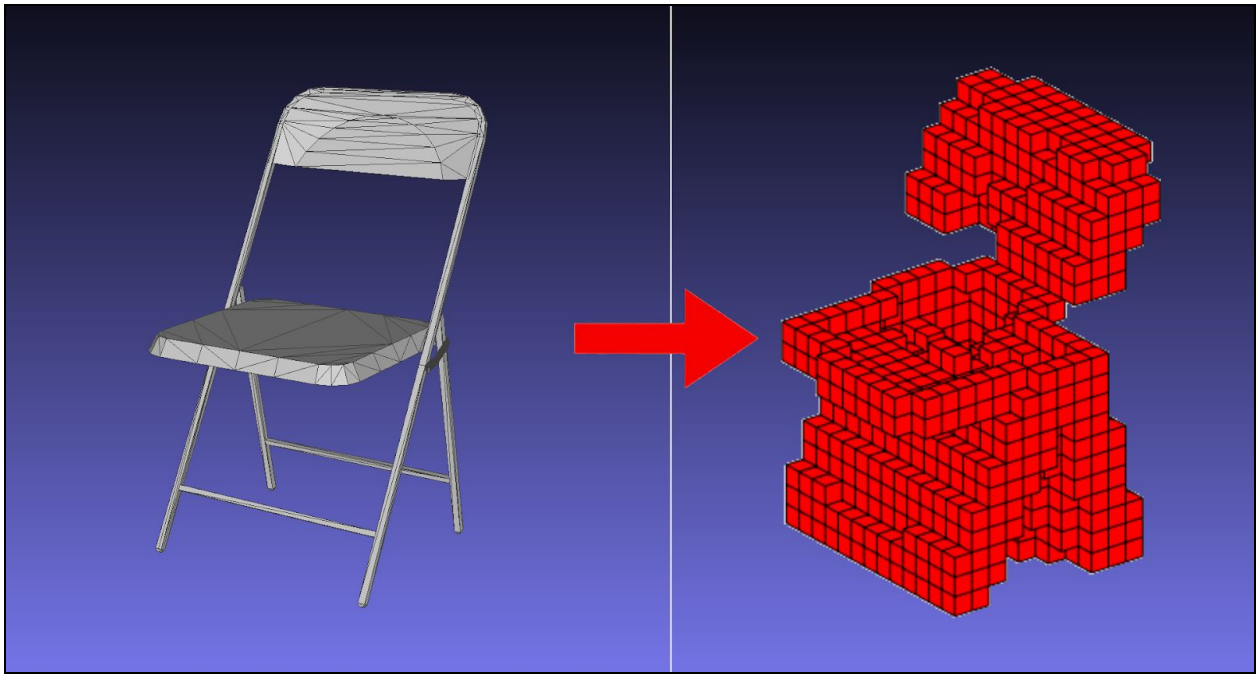


Figure 7: Illustration of Mesh Model Normalized to 30x30x30 Voxel Format

4.2.2 Train the Network

The next step is to execute the training of the network using the parameters in file *run_pretrain.m*. The values of each of the layers determines the number of output maps or features the network would like to generate, how large of a filter size should be used, the type of activation function that should be used, and the length of stride that the filter should be used. The different types of configurations for this file are listed in Section 6.2 Test Configurations. To execute this command, in the MATLAB console use:

```
> run_pretrain.m
```

This step will take a substantial amount of time as the network is being created, initialized with millions of neurons, and also trained as described in Section 6.1. On average, increasing the amount of data by decreasing rotation angle or decreasing stride lengths, training for this took around 4 hours each time.

4.2.3 Test the Convolutional Deep Belief Network

Lastly, the recognition test can be used to see how well the network performs given the training and sample data. The network model must first be loaded into memory so that it can be loaded into a model object. The recognition test is found in *rec_test.m* and can be executed using the following command:

```
> load('pretrained_model.mat')
> rec_test(model)
```

The recognition test is the primary test used to evaluate the network as described in this paper. It simply passes a 3D CAD model through the network and scores how accurate the network was able to classify the object.

5. Tools and Project Configuration

In Section 5, the project hardware configurations are listed to compare vs. existing 3D ShapeNets original configuration.

5.1 Project Hardware

	Original Research Hardware	Current Project
Hardware Name	NVIDIA K40 GPU	NVIDIA GTX 1080 Ti
Core speed	706 MHz	1480 MHz
Architecture	Kepler GK110	Pascal P102
Release Date	Nov 12, 2013	Mar 5th, 2017
Memory Capacity	12288 MB	11264 MB
Memory Speed	1502 MHz	1376 MHz
Memory Bus	384 Bit	352 Bit
Memory Bandwidth	288.4 GB/s	484.48 GB/s
Shader Processing Units	2888	3584
Die size	28 nm	14 nm

Table 3: Hardware comparison between original research and current project

In Table 3 above, the hardware project specifications are listed. The previous 3D ShapeNets project relied on expensive, enterprise level hardware in order to run their experiments [2]. Advancements in technology, however, have made these types of GPU hardware more readily available to average consumers, albeit at a relatively premium cost. The most critical part of the hardware listed above in both cases is the overall memory capacity as a high amounts of memory are required in training the network. In Section 6 below, the total number of nodes are calculated where simple arithmetic operations may be performed on all of

them. With this high memory capacity, the network is able to rapidly process calculations on the many nodes in a highly parallelized fashion.

6. Convolutional Deep Belief Network Training

The following section will discuss the convolutional deep belief network topography as well how the training layers process the input 3D model data. Below is a depiction of the network and a basic description of how it processes the 3D model data on a basic level.

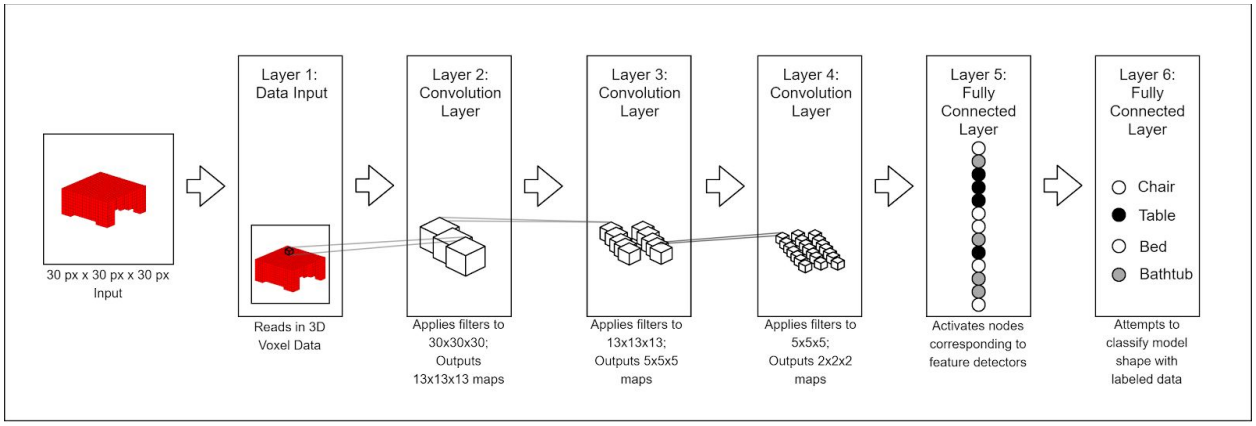


Figure 8: Convolutional Deep Belief Network Recognition Task

6.1 Network Creating and Learning

6.1.1 Layer 1: Data Input

The first step in training the network is to initialize the network nodes each of which possess a weight, an individual bias, and a shared bias whose value is shared between all the nodes of that particular layer. For this project, the default size of each side of the voxel model was 30 voxels, making the dimensions 30 voxels long, 30 voxels wide, and 30 voxels high which were produced when the vector data was transformed into the voxel format. Therefore, naturally, the first initial data layer is simply a 3-dimensional array that is 30 rows, 30 columns, and 30 deep. Each individual element within this array possesses a simple binary value of 1 in the case where the model is filled at a particular position and a value of 0 when that position is

empty. In some cases, a negative value is used to indicate that although the particular index is filled, it is not part of the object.

6.1.2 Layer 2: Convolutional Layer

The following layer, Layer 2 is a convolutional layer and begins to expand the network into many more nodes, using an arbitrary number of output maps or feature extractors similar to those shown in Figure 2 (b). In this project, again the first data input layer has a span of 30 voxels across and the first convolution layer has a kernel size of 6 x 6 x 6 voxels meaning that it will apply a convolution filter across the model using this 6 x 6 x 6 filter. Therefore, the output of the initial data layer alongside the size of the filter would output a 13 x 13 x 13 voxel map by the following formula:

$$l_1 = \frac{(l_0 - f_1)}{s_1} + 1,\ w_1 = \frac{(w_0 - f_1)}{s_1} + 1,\ h_1 = \frac{(h_0 - f_1)}{s_1} + 1$$

$$l_1 = \frac{(30-6)}{2} + 1 = 13,\ w_1 = \frac{(30-6)}{2} + 1 = 13,\ h_1 = \frac{(30-6)}{2} + 1 = 13$$

Therefore, the dimensions of a single map for layer 2, the convolution layer is 13 x 13 x 13. This process is repeated based on the number of output maps that are created, which for this experiment for layer 2 is 48 different output maps. In addition to the output map, each 6 x 6 x 6 voxel filter also randomizes a weight, shared bias, and individual bias which are random values between 0.0 to 1.0.

The total number of nodes created in the second layer is 10,368, given the default project configuration and is based on:

$$nodes_{L2} = outputMaps \times [filtersize]$$

$$nodes_{L2} = 48 \times [6 \times 6 \times 6] = 10{,}368$$

and each node contains a randomly initialized weight between 0.0 and 1.0, but whose value should be very close to zero. The weight values are the essential parameter in which the network will be fine tuned. For example, when passing say a sofa's data through the trained network, the weights will carry the data through the network until the final layer where it determines that it is in fact, a sofa. The randomized initial values of the deep belief network, explained crudely, is so that the network will not always produce the exact same output as previous training runs. This also raises confidence in the values that the network has since it theoretically always converges towards the same point. Additionally each node has an associated shared bias and individual bias. For this project, the randomized weights were calculated by the following formula each time for each of the 10,368 nodes:

$$weight_{L2\,Node} = (x - 0.5) \times 2 \times \sqrt{\left(\frac{6}{k_{L2}{}^3 \times (m_{L2} + p_{L2})}\right)}$$

$$weight_{L2\,Node} = (x - 0.5) \times 2 \times \sqrt{\left(\frac{6}{6^3 \times (48 + 1)}\right)} = (x - 0.5) \times 0.0476$$

where x is a randomized value between 0.0 and 1.0, k is the layer's kernel size, m is the current output map count, and p is the previous layers' output map count. Unlike the weight calculation, the shared and individual bias values are initialized to zero since the 3D voxel models have not yet been analyzed yet. These biases will be increased or decreased based upon calculated gradient values.

The primary purpose of the individual bias directly maps to a single node in a single map. This individual bias is used during the RBM process where it is used to facilitate the reconstruction when the visible units are activated from the current values of the hidden units.

The total number of individual biases are based upon the number of nodes in the previous layer, whereas the total number of shared biases are based upon the current layer's output map size.

$$number\ of\ shared\ bias_{L2} = output\ maps_{L2} = 48$$

For layer 2, the total number of individual biases is 2,700 and the total number of shared biases is 48. The primary purpose of having a shared or channel bias is that the neurons of a single layer should be slightly coupled together such that they are biased in a specific direction together when similar patterns are observed. The number of shared biases directly relates to the number of output maps and therefore are closely coupled on a one-to-one pairing of the output maps.

$$number\ of\ individual\ bias_{L2} = layer\ size_{L1}$$

$$number\ of\ individual\ bias_{L2} = 30 \times 30 \times 30 = 2,700$$

Lastly, the gradient values are also stored for each node, but are similarly all initialized to zero when the network is first created. For each node there are three gradient values correlating to each weight, shared bias, and individual bias whose value changes during the gradient descent process. A simple diagram shows the relationship between weights denoted W, shared biases denoted C, and individual biases denoted B.

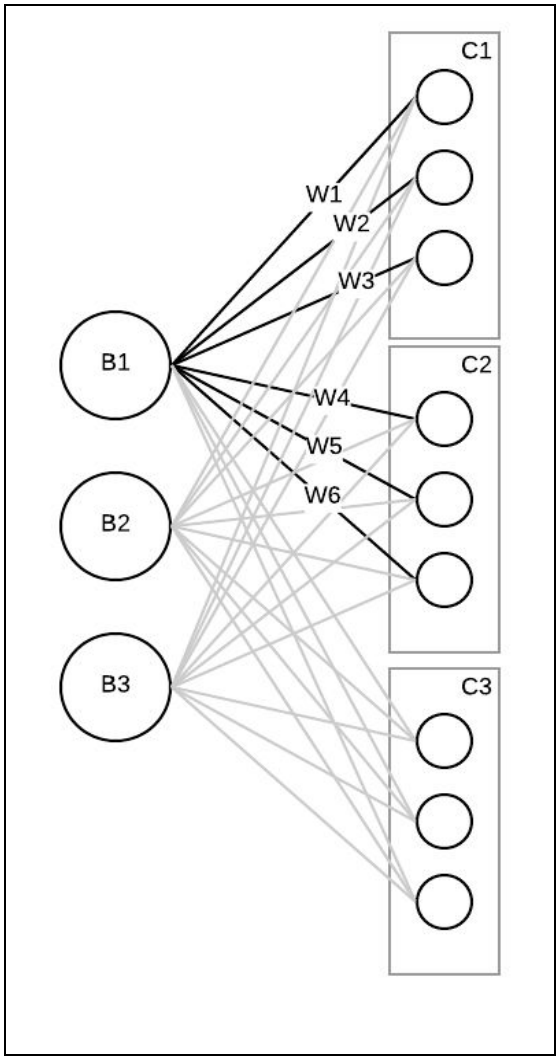


Figure 9: Relationship Diagram between Weights, Shared Bias, and Individual Bias

6.1.3 Layer 3: Convolutional Layer

$$neural\ nodes_{L3} = 160 \times [5 \times 5 \times 5] \times 48 = 960{,}000$$

The third convolutional layer during initialization is similar to that of the previous, Layer 2, and how it is calculated except the total number of nodes significantly expands.

$$weight_{L3\ Node} = (x - 0.5) \times 2 \times \sqrt{\left(\frac{6}{5^3 \times (160 + 48)}\right)} = (x - 0.5) \times 0.03039$$

With the total number of nodes increasing almost 100 times larger than the previous layer, it is important to reiterate the level of parallelism required by GPUs to perform operations on almost 1 million nodes.

$$number\ of\ shared\ bias_{L3} = 160$$

$$number\ of\ individual\ bias_{L3} = 105,456$$

6.1.4 Layer 4: Convolutional Layer

$$neural\ nodes_{L4} = 512 \times [4 \times 4 \times 4] \times 160 = 5{,}242{,}880$$

The fourth convolutional layer contains a huge number of neural nodes reaching over 5 million nodes. The filter size is a smaller, 4 x 4 x 4 voxel shape but are significantly more numerous.

$$weight_{L4\ Node} = (x - 0.5) \times 2 \times \sqrt{\left(\frac{6}{4^3 \times (512 + 160)}\right)} = (x - 0.5) \times 0.02362$$

Moreover, the number of individual biases actually start to decrease since the filter sizes themselves are smaller as the data propagates through the network.

$$number\ of\ shared\ bias_{L4} = 512$$

$$number\ of\ individual\ bias_{L4} = 20,000$$

6.1.5 Layer 5: Fully-connected Layer

The fully connected layers following the convolution layers are significantly smaller as they aggregate the inputs from the convolution layer and classify the object. The first fully connected layer arbitrarily contains 1200 nodes, each of which is connected to all 5,242,880 nodes from the previous convolutional layer. Each of these nodes is either activated or not activated based upon the data input, and upon combining all of these activations determines

which of the 1200 nodes are activated. The combination of activated and non-activated nodes within the 1200 nodes are when the objects become recognizable, but not classifiable without the following layer. The best way to think of this fully connected layer would be: The network is properly able to determine what shape the 3D model has, but it does not know what to call it. This problem is then solved by the final fully-connected layer below.

$$number\ of\ fully\ connected\ nodes_{L5} = 1200$$

6.1.6 Layer 6: Fully-connected Layer + Labeled Data

This fully-connected layer, however, exhibits slightly different behavior than the previous fully-connected layer in that it creates an associative memory with a set of human-labeled data. This labeled data attempts to link together the findings of the 5th fully connected layer and what humans classify those findings as. The number of nodes increases here because the associative memory retains the activations and non activations of individual voxels for labeled shapes such as chairs, desks, etc.

$$number\ of\ fully\ connected\ nodes_{L6} = 4000$$

6.2 Test Configurations

The following tests were run with the hypothesis that increasing the high-level granularity while analyzing the model data would yield a higher recognition accuracy as more higher level features would be given more influence. The first parameter configured for these tests were the stride in which the convolution filters move across the voxelized models. The second parameter configured was rotation angle to examine the additional effects of granularity from the viewport angle in which the 3D model was analyzed from. Moreover, reducing the

incremental angle would increase the amount of training data by double if the angle was reduced by half, for example as more model poses would be taken.

By decreasing the stride distance from 2 to 1, such as in the tests below, the difference for the network would the output map sizes would be changed. In Section 6.1.2, for layer 2, the output map size were 13 x 13 x 13 voxel maps given that the stride was of distance 2 voxels. This, in turn, propagates to layer 3 to have a map size of 5 x 5 x 5 and layer 4 to have a map size of 2 x 2 x 2. The size of the map directly correlates to the total number of individual biases being passed from layer to layer. Modifying the first layer to have a stride of 1, for example, shown in the figure below, would increase the output map size to 24 x 24 x 24 voxels which are not much smaller than the original input of 30 x 30 x 30 voxels. The result of modifying these values slightly increases processing time but drastically increases the amount of memory required in analyzing the models. The upper bound of physical memory 11GB video memory limited the angle to be fixed at 60º when setting all stride values to 1 because 30º increments were running into out of memory exceptions.

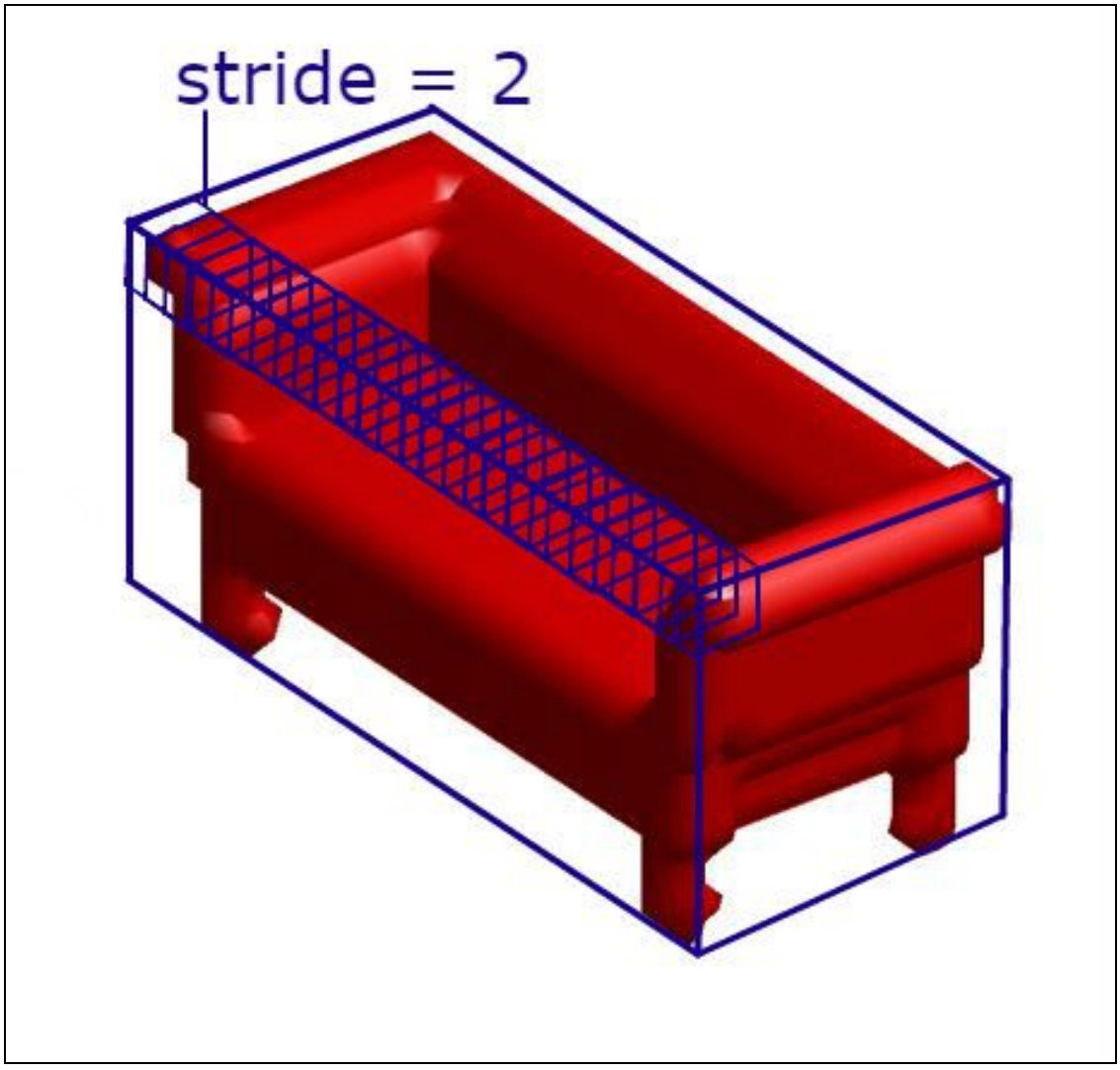


Figure 10: Convolutional Layer 2 with Stride 2

In Figure 10 and Figure 11, a graphical representation of the effects of decreasing the stride distance are shown. Although the change is relatively minimal, its small effect propagates through the Convolutional Deep Belief Network fundamentally changing how it trains and recognizes data.

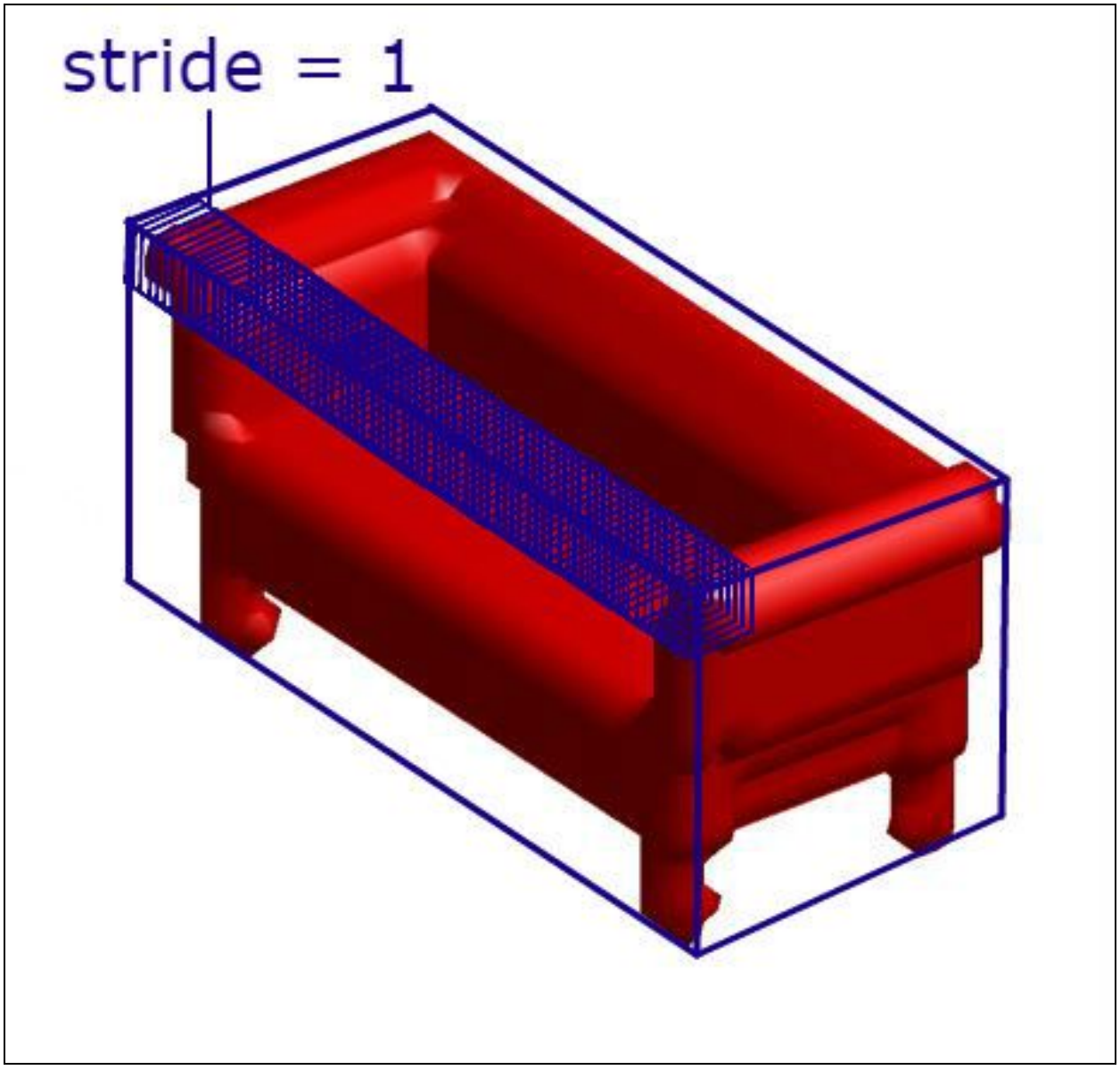


Figure 11: Convolutional Layer 2 with Stride 1

The second parameter modified was to change the rotation angle of the virtual camera, which the 3D model training data used to capture poses. The default angle of rotation about the figure was 30º and incrementally decreased and increased to determine if it had a correlation to recognition accuracy. By decreasing the angle to 15º, the training model read in double the number of poses for each of the models, from 12 to 24. By increasing the angle to 45º, the

training model poses decreases from 12 to 8. This incremental process was continued for 60º, 75º and 90º as well. The importance of pose in this experiment was that the angle of each pose may expose some previously occluded component. Below is a depiction of rotating a chair model in 45º increments. For the training of the network, however, these mesh models are converted into voxel models such as in Figure 7 which are then used for data input into the network.

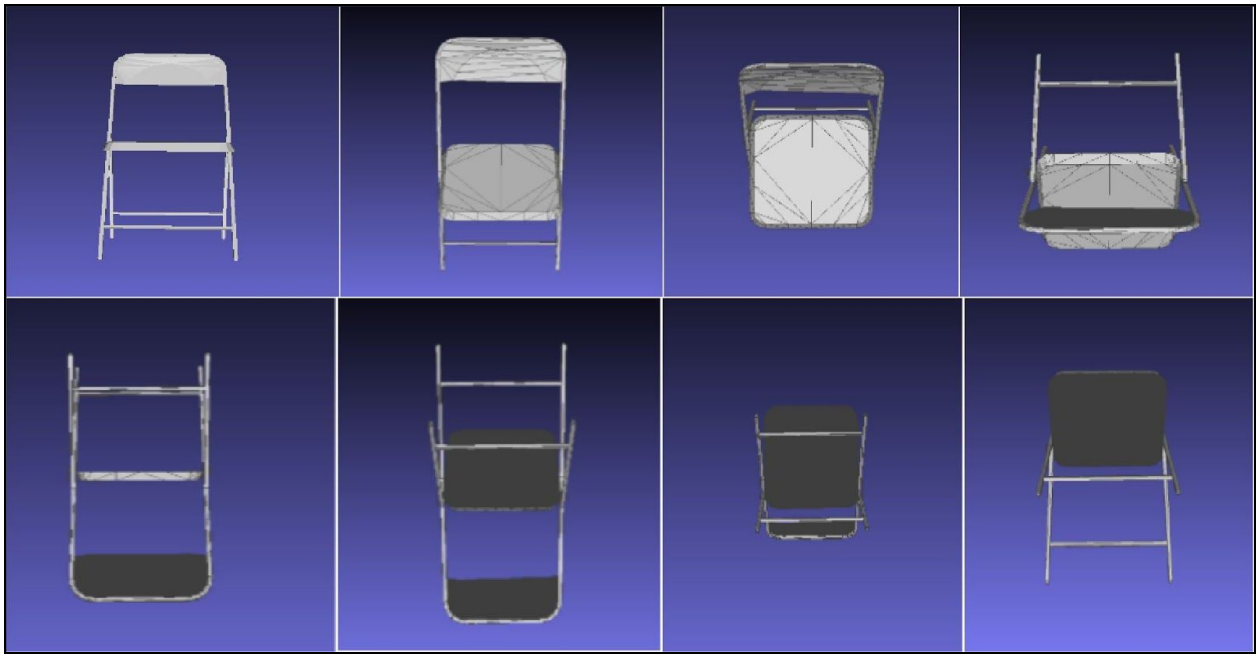


Figure 12: Poses for Chair Mesh Model in 45º Increments for Training

6.2.1 Network Configuration Parameters for Tests

	Training Dimensions			Convolutional Layer 2			Convolutional Layer 3			Convolutional Layer 4			Fully Connected Layer 5		Fully Connected Layer 6	
Test #	Object Size (px x px x px)	Padding Size (px)	**Rotation Angle (°)**	Filters	Filter Size (px)	**Stride Length (px)**	Filters	Filter Size (px)	**Stride Length (px)**	Filters	Filter Size (px)	**Stride Length (px)**	Nodes	Activation Function	Nodes	Activation Function
1	24x24x24	3	**15**	48	6x6x6	**2**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
2	24x24x24	3	**15**	48	6x6x6	**2**	160	5x5x5	**1**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
3	24x24x24	3	**15**	48	6x6x6	**1**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
4	24x24x24	3	**30**	48	6x6x6	**2**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
5	24x24x24	3	**30**	48	6x6x6	**2**	160	5x5x5	**1**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
6	24x24x24	3	**30**	48	6x6x6	**1**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
7	24x24x24	3	**45**	48	6x6x6	**2**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
8	24x24x24	3	**45**	48	6x6x6	**2**	160	5x5x5	**1**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
9	24x24x24	3	**45**	48	6x6x6	**1**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
10	24x24x24	3	**60**	48	6x6x6	**2**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
11	24x24x24	3	**60**	48	6x6x6	**2**	160	5x5x5	**1**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
12	24x24x24	3	**60**	48	6x6x6	**1**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
13	24x24x24	3	**75**	48	6x6x6	**2**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
14	24x24x24	3	**75**	48	6x6x6	**2**	160	5x5x5	**1**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
15	24x24x24	3	**75**	48	6x6x6	**1**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
16	24x24x24	3	**90**	48	6x6x6	**2**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
17	24x24x24	3	**90**	48	6x6x6	**2**	160	5x5x5	**1**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid
18	24x24x24	3	**90**	48	6x6x6	**1**	160	5x5x5	**2**	512	4x4x4	**1**	1200	Sigmoid	4000	Sigmoid

Table 4. Convolutional Deep Belief Network Configuration Tests

In the table above, the 18 tests that were conducted during this project are listed. Each row illustrates a set of slightly different parameters that configure the network and how it trains and recognizes different models. As specified in Section 6.2, the two critical parameters for this experiment are rotation angle and stride length, emphasized in bold in Table 4. The remaining parameters can be modified as well, but remain fixed in order to reduce any influence they may have on the test results. By default, the stride length for convolution layer 2 is 2 and when changed to 1, this highlights the higher level feature details such as curves, corners or geometric fundamental shapes. Conversely, decreasing the stride length for convolution layer 3 increases the emphasis on more abstract patterns that humans may not be able to differentiate such as in the example in Figure 2.

7. Results

A brief summary of the test results from the experiment are shown below in Section 7.1 listing the input parameters, recognition results and network training time. A further analysis of these results follows in Section 7.2.

7.1 Test Results

Run #	Rotation Angle	Layer 2 Stride Distance	Layer 3 Stride Distance	Layer 4 Stride Distance	Model Voxelation & Training Duration (hh:mm:ss)	Recognition Accuracy
1	15º	2	2	1	2:12:09	42.38%
2	15º	2	1	1	2:31:18	38.75%
3	15º	1	2	1	3:31:20	37.68%
4 (Default)	30º	2	2	1	1:04:45	47.63%
5	30º	2	1	1	1:39:34	52.92%
6	30º	1	2	1	2:23:08	37.58%
7	45º	2	2	1	1:03:12	56.46%
8	45º	2	1	1	1:22:25	50.45%
9	45º	1	2	1	2:05:26	49.79%
10	60º	2	2	1	0:58:43	**60.54%**
11	60º	2	1	1	1:36:17	59.33%
12	60º	1	2	1	1:54:04	58.58%
13	75º	2	2	1	0:57:18	**59.83%**
14	75º	2	1	1	1:17:11	57.92%
15	75º	1	2	1	1:59:03	58.60%
16	90º	2	2	1	1:06:43	**60.84%**
17	90º	2	1	1	1:25:38	53.56%
18	90º	1	2	1	2:05:45	57.33%

Table 5. Test Results

7.2 Results Analysis

7.2.1 Rotation Angle vs. Recognition Accuracy

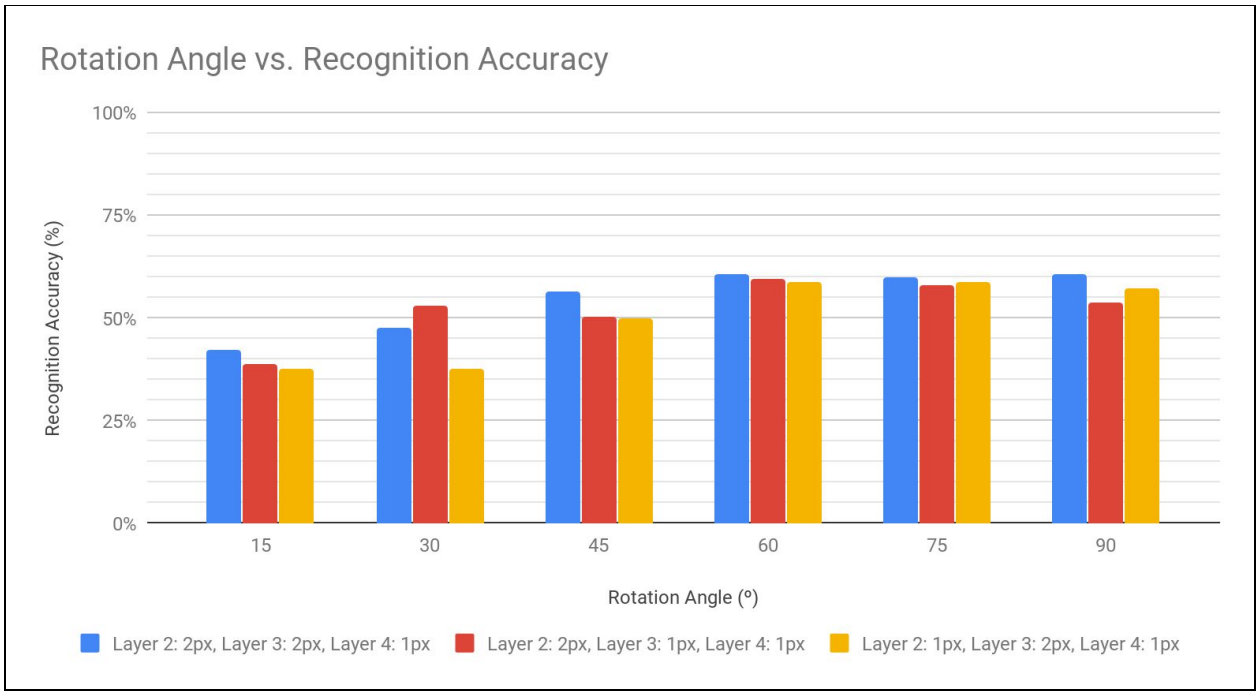


Chart 4: Rotation Angle vs. Recognition Accuracy

In Chart 4 above, the graph illustrates the effects of different angles of rotation during the training process of the Convolutional Deep Belief Network. Additionally, as noted in the legend at the bottom of the chart, columns are additionally grouped by the Layer 2, Layer 3, and Layer 4 configurations. The default rotation angle of the initial project conducted was 30º and is used as a baseline when comparing to the rest. As mentioned in Section 6.2, the number of poses taken for each model is 360º divided by the angle increment of 30º, resulting in 12 poses captured for each model within the 10 different shape categories. Also mentioned previously, reducing this angle to 15º effectively doubles the number of poses taken for each individual model.

Although more poses are captured for each individual model during the training process, the results actually indicate an adverse effect which may have potentially been caused by two

possible factors: (1) overfitting or (2) possibly shape malformation during the training process [18]. With a rotation angle of 15º, the possibility of overfitting may have increased as each particular shape, such as a chair for example, is now captured in 24 different poses for each chair data model. After the training process, each different pose may have understated the more fundamental shape of the chair, resulting in more ambiguity when trying to classify the 3D model shape. The second possible explanation of a reduction in recognition accuracy is shape malformation. During the voxelation process the scaled down voxel models of each of the shape inputs may not scale accurately such as the case depicted in Figure 7. The problem is due to the limitation of the 30 x 30 x 30 voxel volume of each of the figures.

The overall result illustrated in Chart 4, is that the recognition accuracy increases as the rotation angle increments increase up to a certain point. The chart would suggest that after reaching a rotation angle of 60º, any increase did not improve recognition accuracy and seemed to saturate at around 60%. This result would suggest that fewer poses taken for each of the data shapes when turned into voxelated models increases recognition due to the fact that the overall shapes are a strong enough factor in classification of a shape. This result may be different if a larger number of shape categories were included, but for the 10 shape categories listed in this project, the recognition accuracy seemed to maximize at 60%.

7.2.2 Rotation Angle vs. Training Time

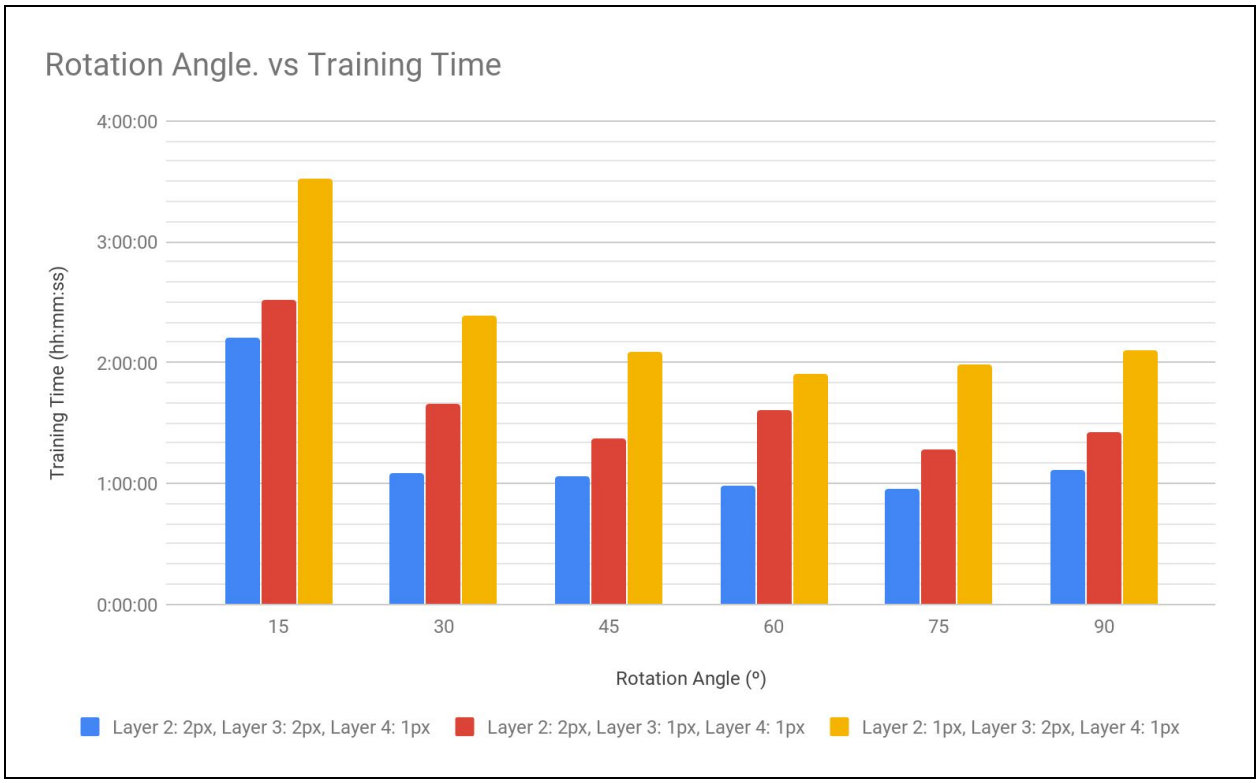


Chart 5: Rotation Angle vs. Training Time

The results mentioned in the previous section above relating to rotation angle and accuracy, showed a positive correlation in accuracy when increasing the angle increment. In Chart 5 above, a similar analysis was done on exactly how long it took to create the voxelized format, train the network, and run recognition tasks. Again, using 30º as a baseline, decreasing the angle increment to 15º resulted in almost double the amount of network training time in some cases required for the base layer configuration. However, increasing the rotation angle to above 30º decreased the amount of training time required which is both expected and favorable to the accuracy results. The reason as to why this reduction is expected is due to the reduction of poses for each input data shape.

7.2.3 Stride Distance vs. Recognition Accuracy

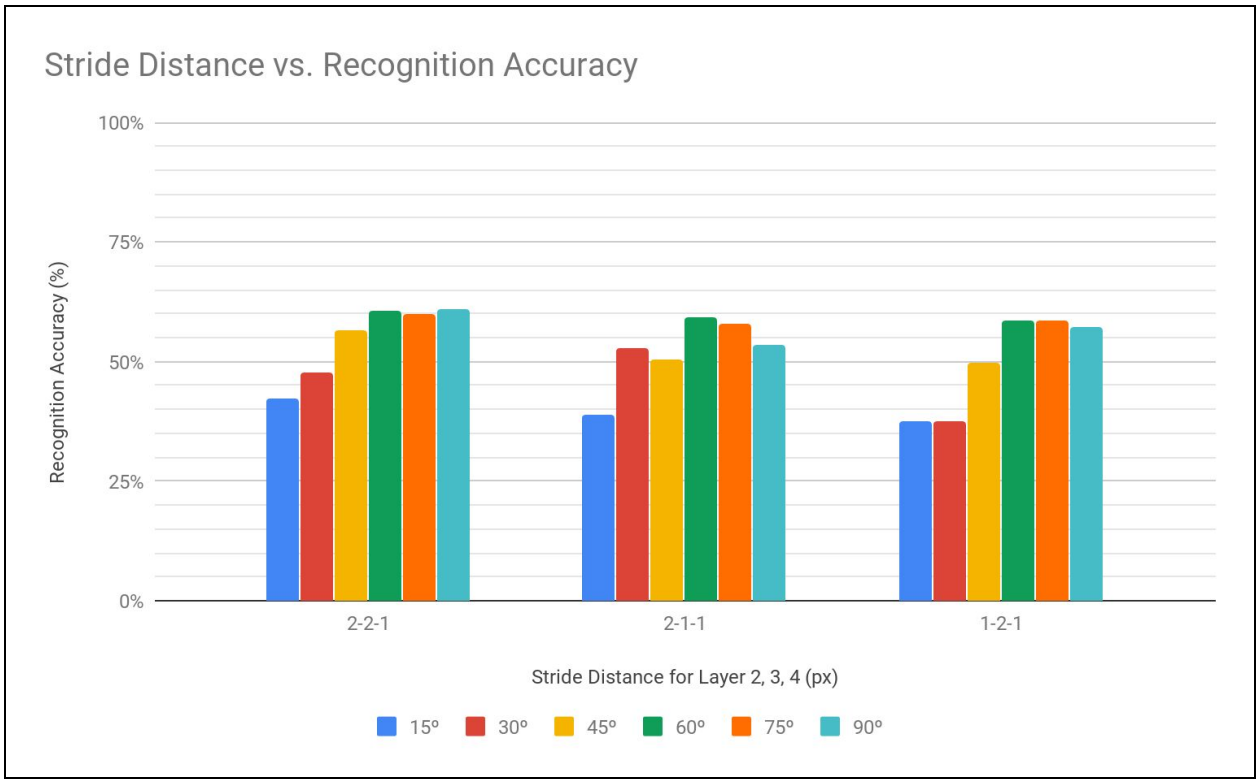


Chart 6: Stride Distance vs. Recognition Accuracy

Although there seemed to be a significant correlation to accuracy and rotation angle, different stride distances in the convolutional layer filters seem to have a significantly dampened effect. The third set of columns in Chart 6 indicate the case where the Convolutional Layer 2 performs a smaller change in distance in each step of the filters when training the network. This would suggest that the stride distance of 2 versus a stride distance of 1 does not miss any important cues as the filter traverses the model. This would also suggest that this granular traversal across the model in this project led to higher ambiguity in the classification of different shapes perhaps due to an overemphasis of higher level versus lower level cues.

7.2.4 Stride Distance vs. Training Time

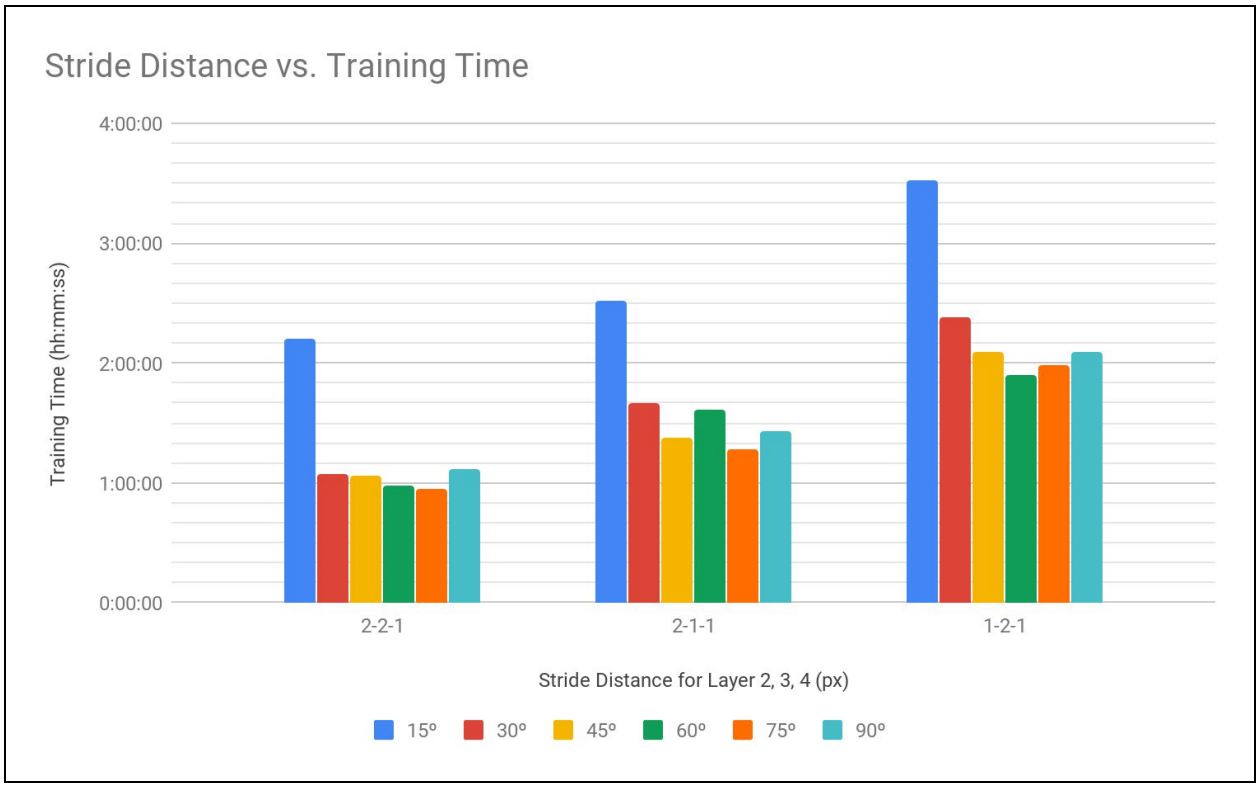


Chart 7: Stride Distance vs. Training Time

Similar to the analysis done in 7.2.2, some supplemental information for the relationship between decreasing stride distance versus training time is shown above. The results above are simply as expected: The more granular traversal the network performs on individual models and shapes, the more time is required in order to process the data and train the network. In the third set of columns in Chart 7, the Layer 2: 1 px, Layer 3: 2 px, Layer 4: 1 px configuration took substantially longer than the other configurations for training the network. This was due to the fact that in the first convolutional layer, Layer 2, the resultant output map size, which is divided by the stride length, was increased in this early phase. This increase in output map size propagates through the following layers, increasing the total input that each layer needs to process.

8. Conclusion

Altogether, attempting to replicate an existing project using consumer-grade hardware for training a deep neural network proved to be far more complicated than originally thought. However, the amount of research, and knowledge gained in both understanding how neural networks, convolutional neural networks, deep belief networks, and convolutional deep belief networks was thoroughly insightful. Although it was difficult, the core requirements of developing the network were able to be met, the network was able to be trained, it was able to recognize and classify 3D model shapes and even potential methods of improvement were discovered.

The test results in Section 7.1 showed a significant improvement from the base, increasing recognition accuracy from 47.63% to as high as 60.84%. Although it is difficult to determine why exactly features are better recognized by a convolutional deep belief network, the tests that were run showed an empirical correlation to the overall improvement in shape recognition. Moreover, the tests results also confirmed the expectation that the overall training time should be less when training the network with fewer poses for each shape model. One relationship shown by running these tests are that a high level of detail is not necessarily important when trying to classify shapes, up to a certain extent.

The more unfavorable result was shown in the negative correlation between stride distance and overall shape recognition, which opposed the initial assumption. Similar to the correlation above with capturing fewer poses per shape, the reduced stride value increases the level of detail. And again, the data would suggest that such a granular level of voxel analysis is not required and may in fact cause overfitting, reducing recognition accuracy.

9. Future Work

One important fact to note is that this network was only configured, due to sample limitations and feasible training time, to learn about 10 different shape classes: bathtub, bed, chair, desk, dresser, monitor, night stand, sofa, table, and toilet. There are clearly a much larger amount of possible shapes such as cars, planes, lamps, etc. but much of those 3D model CAD data is not easily available or simply do not exist. Therefore, for future work, the number of classes if available should expand as it may actually produce even further insightful results. This project has shown that Convolutional Deep Belief Networks are capable of handling 3D model data and as hardware continues to evolve, so can their capabilities.

One limitation of this project was the voxelized sizes of the data models. The initial models are object file format (OFF) files who possess a much higher level of precision in a triangular mesh format. It would also be of interest to examine the effects of significantly increasing the model voxelations beyond 30 x 30 x 30 px and perhaps reach up to over 100 x 100 x 100 px. This would, of course, require additional training time, require more memory, and perhaps even require additional convolutional layers when creating the much smaller size feature extractors.

10. References

1. S. Song and J. Xiao. Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, 2016, pp. 808-816.
2. Z. Wu, et al. 3D Shapenets: A Deep Representation for Volumetric Shapes. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1912-1920. https://github.com/zhirongw/3DShapeNets
3. X. Peng, B. Sun, K. Ali and K. Saenko. Learning Deep Object Detectors from 3D Models. *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, 2015, pp. 1278-1286.
4. X. Xu and S. Todorovic. Beam search for learning a deep Convolutional Neural Network of 3D shapes. *23rd International Conference on Pattern Recognition (ICPR)*. Cancun, 2016, pp. 3506-3511.
5. A. Zeng, et al. 3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017
6. X. Wang, R. Li and J. Currey. Leveraging 2D and 3D cues for fine-grained object classification. *2016 IEEE International Conference on Image Processing (ICIP)*. Phoenix, AZ, 2016, pp. 1354-1358.
7. S. Richard, et al. Convolutional-recursive deep learning for 3D object classification. *Advances in Neural Information Processing Systems*. 2012.
8. D. B. Kirk, W. H. Hwu. Programming Massively Parallel Processors: A Hands-on Approach. Elsevier Inc. Cambridge, MA. 2017, pp 149-174.
9. A. Chang, et al. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012*. 2015
10. G. E. Hinton (2009) Deep belief networks. Scholarpedia, 4(5):5947., revision #91189
11. G. E. Hinton (2007) UCL Tutorial on: Deep belief nets [PDF Document]. Retrieved from http://www.cs.toronto.edu/~hinton/ucltutorial.pdf

12. F. Wu, et. al. Regularized Deep Belief Network for Image Attribution Detection. *IEEE Transactions on Circuits and Systems for Video Technology, Vol 27(7)*. 2017

13. C. R. Chen, G. G. C. Lee, Y. Xia, W. S. Lin, T. Suzumura and C. Lin, "Efficient Multi-training Framework of Image Deep Learning on GPU Cluster," *2015 IEEE International Symposium on Multimedia (ISM)*, Miami, FL, 2015, pp. 489-494.

14. J. Wu, et al. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. *arXiv: 1610.07584v2.* 2017

15. D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *In Proceedings of the 27th International Conference on Neural Information Processing Systems - Vol(2),* 2014, pp. 2366-2374.

16. F. Liu, C. Shen, G. Lin, I. Reid. "Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields" in *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 38, no. 10, pp. 2024-2039, 2016.

17. R. Socher, B. Huval, B. Bhat, C. D. Manning, and A. Y. Ng. Convolutional-recursive deep learning for 3D object classification. *In Proceedings of the 25th International Conference on Neural Information Processing Systems - Vol(1)*. 2012. pp 656-664.

18. Y. Zheng, D. Liu., B. Georgescu., H. Nguyen., D. Comaniciu. 3D Deep Learning for Efficient and Robust Landmark Detection in Volumetric Data. *MICCAI*. 2015.

19. J. Wu, C. Zhang,, X. Zhang, Z. Zhang, W. T. Freeman, & J.B. Tenenbaum. Learning Shape Priors for Single-View 3D Completion And Reconstruction. *ECCV*. 2018.

20. J. Wu, Y. Wang, T. Xue, X. Sun, W. T. Freeman, J. B Tenenbaum. (2017). MarrNet: 3D Shape Reconstruction via 2.5D Sketches. *arXiv:1711.03129.*